ATP/HPO Installation and Operation

ATP/HPO

Version 1.4

MVS Operating Environment

PassGo

Publication number PAT0001.012

Published by:

PassGo Technologies Ltd, Horton Manor, Ilminster, Somerset, TA19 9PY, England.

Publication number PAT0001.012

Twelth Edition (September 2001)

Changes will be made periodically to the information contained in this book. If your book does not accurately reflect the level of product you are using, this may be due to a fix being applied to the product which has still to be released as a book update.

Preface

Prerequisites for running ATP/HPO

ATP/HPO has been designed to run under the latest releases of MVS, MVS/XA and MVS/ESA running ACF/VTAM V3.3 and earlier, in both SNA and NON-SNA environments.

Devices supported by ATP/HPO

- all models of 317x, 318x, 319x and 327x (or their equivalents), utilizing the maximum available screen display area
- the 3290 Information Panel with each interactive panel in 3270 mode
- LUTYPE 1 printers (SNA printers with the SCS feature)
- devices attached through NTO
- IBM 3767 and compatible devices

Application sessions using MULTSESS/HPO are supported in the terminal's native mode.

Terminal to MULTSESS/HPO sessions are supported in model 2 mode.

About the ATP/HPO Installation and Operation Guide

This installation and operation guide is for use by technical personnel who are going to install, customize and use ATP/HPO.

The contents of this guide should be read before installation.

Related publications

The MULTSESS/HPO User Installation Manual describes how to install MULTSESS/HPO in your environment.

The MULTSESS/HPO User Reference Manual contains details of commands that the user may enter at the terminal and error messages that may be issued to the terminal.

The MULTSESS/HPO User Reference Manual should be read in conjunction with the MULTSESS/HPO User Installation Manual when certain installation tasks are being performed, such as defining command sequences in the user profile dataset, and may be used as a stand-alone reference work by terminal users.

The MULTSESS/HPO Technical Reference and Customization Reference manuals provide the necessary technical background and customization procedures for running MULTSESS/HPO.

Future Publication changes

Changes will be made periodically to this publication to reflect new releases and facilities. When this occurs you will be supplied with update pages or a new updated manual.

Reader's comment form

A form for the reader's comments is provided at the back of this manual.

Any information supplied may be used or distributed by the authors in any manner considered appropriate without incurring any obligation whatsoever.

Table of contents

This page intentionally left blank

Chapter 1 - Introduction to ATP/HPO

What is ATP/HPO?

ATP is an advanced VTAM application allowing, in the form of scripts, the pre-coding of statements that emulate keyboard functions.

ATP eliminates the keying of repetitive terminal-based functions. The MULTSESS/HPO user need only invoke an ATP script to automatically perform any number of inputs.

Scripts may include decision making functions based on data from the application or variable data supplied by the MULTSESS/HPO user to provide alternative navigation through the script, or to detect abnormal conditions and take corrective action.

Interfaces to MULTSESS/HPO

ATP V1.4 interfaces to MULTSESS/HPO V1.1 to provide a virtual user concept to automate common or repetitive transactions to any VTAM application, eg logon to CICS, select a transaction from the CICS menu and initiate the transaction, all via a single keystroke. The end user sees only the MULTSESS/HPO menu followed by the CICS transaction screen, drastically reducing the network traffic and overhead and increasing end-user productivity.

Major features of ATP/HPO

- No limit to the number of users running scripts
- A user can have many scripts running concurrently
- Background scripts can run concurrent with normal terminal activity
- Automatic script invocation at MULTSESS/HPO logon
- Automatic script invocation at application selection time
- Protection against unauthorized use of scripts
- Shared or exclusive script datasets
- Controllable environment
- Does not affect the number of MULTSESS/HPO users
- Provides script support for multiple MULTSESS/HPO address spaces

This page intentionally left blank

Chapter 2 - Installation overview

1. **Review the distribution material**

   Accompanying this manual you should find an Installation Guide and the product installation tape. If you have not received either of these, contact your local support office before attempting the installation process.

   You are strongly advised to review the whole of this chapter and the Installation Guide before commencing installation.

   Refer also to the following parts of the MULTSESS/HPO User Installation Manual:

   Chapter 2 - VTAM updates

   Chapter 3 - Defining applications to MULTSESS/HPO - SCRIPT statement

   Chapter 5 - Pre-startup and startup options - ATP-MAXIMUM-RUSIZE

   Chapter 7 - Creating the ATP/HPO control table

2. **Load the distribution libraries**

   The steps required to load the distribution tape are described in the accompanying Installation Guide.

3. **Define ATP to VTAM**

   The following VTAM changes are required:

   - an application entry for ATP in SYS1.VTAMLST
   - If ATP is cross domain, an entry in the CDRSC table in the other domains.

4. **Startup options**

   Review the possible startup parameters as described in *Startup options* on page 2.2.

5. **JCL to run ATP**

   Create the JCL required to run ATP as described in *JCL to run ATP/HPO* on page 2.5.

Loading the distribution tape

Installation Guide
The Installation Guide accompanying the distribution tape lists the tape contents and describes the steps required to install ATP on your system.

Startup options

The following parameters must be specified via the ATPPARM DD statement in the JCL used to run ATP/HPO.

ACBNAME=*name*

Used to indicate to ATP the ACBNAME to use during execution. This parameter is mandatory.

ATTRIBUTE=BLANK

Specifies that attribute characters in the outbound datastream are to be treated as blanks (X'40') when testing for data strings in an IF, JUMPIF, UNTIL or WHILE script statement.

CODE=*aaaaaaaaa*

Defines the authorization code, based on the CPU serial number, enabling ATP to run as a licensed product. There is no default, the code can only be obtained from your Support Office.

DATE=*format*

This controls the date format on the ATPLOG. Both the field order and separation characters can be specified, for example:

    YY:DD:MM

Default is DD/MM/YY, day, month and year separated by the slash (/) character.

Note: In compliance with year 2000 requirements, displayed dates now have a four digit year format. The format of the DATE= startup option remains unchanged, ie the year format is a two digit entry (YY) which is changed to four digits (YYYY) for display.

LIBRARY-PREFIX=*prefix*

This is the leading part of the dataset name used to build the name of a private script library. Specify a valid operating system dataset name qualifier. The maximum length is 35 characters if the LIBRARY-SUFFIX parameter is omitted, otherwise the combined length of prefix and suffix must not exceed 34 characters. Refer to *Script libraries* on page 2.7 for further details.

LIBRARY-SUFFIX=*suffix*

This is the trailing portion of the dataset name used to build the name of a private script library. Specify a valid operating system dataset name suffix. The maximum length is 35 characters if the LIBRARY-PREFIX parameter is omitted, otherwise the combined length of prefix and suffix must not exceed 34 characters. Refer to *Script libraries* on page 2.7 for further details.

LOG-PAGE-LENGTH=*nn*

Specifies the number of lines per page, maximum 99, that ATP will write when producing the log of events. If omitted, the default is 60 lines per page.

MAXGOTO=*nn*

Specifies the maximum number of GOTO statements that can be processed consecutively without any terminal I/O before the script is terminated. The value must be numeric in the range 1 to 999999. This value should be set low enough to avoid the possibility of a SCRIPT looping.

The default is 20.

MAXIMUM-RUSIZE=*nn*K|*nnnn*B

This controls the size of the largest REQUEST UNIT (RU), specified in bytes (B) or kilobytes (K), that ATP/HPO can receive from a calling MULTSESS/HPO. If specified too large, virtual storage will be wasted. If defined too small, a connection attempt will be refused. The value specified should be equal to the largest value coded as the ATP-MAXIMUM-RUSIZE startup option for any MULTSESS/HPO that will communicate with this ATP/HPO address space. The value should not exceed 64K.

The default is 5K.

MAXIMUM-STATEMENTS=*nnnn*

nnnn specifies the maximum number of statements that will be executed before ATP/HPO assumes an error condition has occurred (for example a loop) and terminates execution of the script. The value should be set high enough to prevent premature termination of valid scripts and low enough to detect an error condition. The value may not exceed 99999.

The default is 99999.

MAXIMUM-SUBTASKS=*nn*

The value can be from 1 to 240 and defines the maximum number of concurrently active I/O subtasks for loading scripts into storage. When all subtasks are in use, requests to load new scripts will be queued until a subtask becomes available.

The default is 10.

MAXIMUM-VARIABLES=*nn*

The maximum number of variables per script that ATP/HPO is to allow. Refer to *Using variables in scripts* on page 2.9 for further details.

The default is 10.

NULL=BLANK

Specifies that nulls (X'00') in the outbound datastream are to be treated as blanks (X'40') when testing for data strings in an IF, JUMPIF, UNTIL or WHILE script statement.

PASSWORD=*acb-passwd*

Allows the specification of a password for the ACB that ATP/HPO runs under.

PUBLIC-LIBRARY=*dataset-name*

The fully qualified name of the library where public (unrestricted) scripts are stored. Specify a valid operating system dataset name up to a maximum of 44 characters. Refer to *Script libraries* on page 2.7 for further details.

PERMANENT-SCRIPT=*script/library,script/library,....*

Tells ATP which scripts are to be permanently resident in storage. If library is omitted, the public library is assumed. This statement may be coded multiple times. Refer to *Script management* on page 2.6 for further details.

SCRIPT-STORAGE=*nnnn*B|*nn*K

The maximum amount of virtual storage ATP/HPO is to use for storing Scripts. Specify a value in bytes (B) or kilobytes (K).

The default is 1K.

STOP/MODIFY=YES|NO

If YES is specified, operator commands can be entered from the system console using the MVS stop (P) and modify (F) commands. Refer to *System operator interface* on page 2.10 for further details.

TEST=YES|NO

Specify YES to cause ATP to always load a fresh copy of a script from disk each time the script is invoked. An existing in-storage version of the script (including scripts marked as permanently resident) will be deleted. Refer to *Script management* on page 2.6 for further details.

The default is NO.

TRACE=ON|OFF|ALL

Dependent on the setting of this option, ATP/HPO VTAM tracing routines will be invoked. This will produce large amounts of output, cause some overhead and should only be used on the advice of your local support office.

The default is OFF.

JCL to run ATP/HPO

The JCL to run ATP/HPO is defined below. It is recommended that you run it as a started task.

Example

```
//ATP    PROC OPT+STARTOPT
//****************************************
//*                                      *
//*    THIS PROCEDURE EXECUTES ATP       *
//*                                      *
//****************************************
//AT10    EXEC PGM=ATP,REGION=1024K
//STEPLIB  DD DSN=CAK0001,ATPV10.LOAD,DISP=SHR
//ATPLOG   DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//ATPPARM  DD DSN=CAK0001.ATPV10.CNTL(&OPT).DISP=SHR
```

Note: A sample set of startup options and started task JCL is supplied in the control library loaded from the distribution tape.

For further details, please refer to the *Installation Guide* that accompanies the tape.

Script management

Script libraries
Scripts are defined as members of a script library, a standard partitioned dataset. ATP/HPO supports multiple script libraries to assist in script management and security.

Script statements
Script statements are stored as plain language statements, as described in *Chapter 4 - ATP/HPO language statements*. **No assemblies or other preprocessing are required.**

Script execution
When execution of a script is requested, ATP/HPO first checks whether the required script is already resident in storage. If so and the script has not been marked for deletion by a RESET command, the in-storage copy is used. Otherwise an I/O subtask is invoked to read the script from the appropriate script library and convert it to compressed internal format.

Initial script development
During initial script development when frequent changes are made to script library members, it is desirable to have ATP/HPO **always** load a new copy of the script from the library without the need for a RESET command. This is achieved by specifying the TEST=YES startup option. Use of this option involves extra overhead and will increase response times. Remember to revert to TEST=NO when script development is complete.

I/O subtasks
The number of I/O subtasks that can be concurrently active may be limited by the MAXIMUM-SUBTASKS= startup option. If all available subtasks are in use, outstanding script load requests are queued pending completion of a current load operation.

Script storage
The amount of virtual storage available to hold in-storage copies of scripts may be limited by specifying the SCRIPT-STORAGE= startup option. If the amount of available storage is insufficient to load the new script, ATP/HPO searches for scripts to delete from storage to make space for the new script. Scripts that have been reset and whose use count has dropped to zero are primary candidates for deletion from storage, followed by scripts that have the oldest time of last reference. Scripts that are in use are never considered for deletion, even if they have been reset.

Scripts may be marked as permanently resident to prevent them ever being removed from storage except as the result of a RESET command. The startup option PERMANENT-SCRIPT= may be specified as many times as required to define scripts that are never to be considered for deletion from storage.

Script libraries

Script storage

ATP/HPO allows scripts to be stored in an unlimited number of different script libraries. This enables maintenance to be conveniently devolved and managed by different departments and can provide security for scripts that invoke sensitive functions, such as access to payroll data.

The public library

ATP must always have access to at least one library, the public library. The fully qualified dataset name of this library must be supplied in the PUBLIC-LIBRARY= startup parameter. Unless MULTSESS/HPO passes an alternative library identifier as part of the script request, all scripts will be called from the public library.

Private libraries

Access to private libraries is invoked by a combination of ATP/HPO startup options and information which MULTSESS/HPO obtains from its own ATP/HPO Control Table and which is passed to ATP/HPO as part of the initial request to run a script for a MULTSESS/HPO user.

ATP/HPO control table

The MULTSESS ATP/HPO Control Table is used by MULTSESS/HPO to verify that a user is authorized to run the requested script on his particular session and to obtain the library identifier from which the script is to be loaded. Since a single ATP/HPO address space may service script requests from multiple copies of MULTSESS/HPO, having a separate ATP control table for each MULTSESS/HPO address space enables different ATP/HPO security information to be maintained independently for separate user groups.

Starting a script

When MULTSESS/HPO asks ATP/HPO to start a script, the library id (if any) is passed as part of the request. If no library id is passed, ATP/HPO uses its public script library, otherwise the library id is used to build a fully qualified dataset name in conjunction with the LIBRARY-PREFIX and LIBRARY-SUFFIX startup options. The values specified for these parameters are 'wrapped around' the library id passed by MULTSESS/HPO to form the script library name, for example:

```
LIBRARY-PREFIX=SYS1                 )
MULTSESS gives PAYROLL              )==>SYS1.PAYROLL.SCRIPTS
LIBRARY-SUFFIX=SCRIPTS              )

LIBRARY-PREFIX=ATP.SCRIPTS          )
MULTSESS gives PENSIONS             )==>ATP.SCRIPTS.PENSIONS
LIBRARY-SUFFIX parameter omitted    )
```

Script security

The use of the ATP/HPO Control Table for script security is described in *Chapter 7 - Creating the ATP/HPO control table* in the MULTSESS/HPO User Installation Manual.

Note: Where scripts of the same name exist in more than one library, ATP/HPO always differentiates between in-storage copies and treats them as separate scripts. Requests to run a script, or to reset a script, always take account of the associated library id in order to maintain security and integrity.

Invoking scripts

Scripts may be invoked on behalf of a MULTSESS/HPO user:

- automatically at session initiation (logon script)
- at any time during the life of a session
- automatically at session termination (logoff script)

Logon scripts
A logon script may run automatically when a session is started as the result of:

- the SCRIPT= parameter in the ADT (Application Definition Table) description of the application. The named script will be run as part of session initiation for all users requesting that symbolic application name. A sample ADT is provided in *Chapter 3 - Defining applications to MULTSESS/HPO* in the MULTSESS/HPO User Installation Manual.
- the script parameter on the appropriate SESSION statement in a user's MULTSESS/HPO directory entry. The script will be run as part of session initiation for this user. Sample directory entries are provided in *Chapter 4 - Defining users to MULTSESS/HPO* in the MULTSESS/HPO User Installation Manual.
- the script parameter on a SESSION command, either typed at the terminal or included in the user's profile member. The SESSION command is described in the MULTSESS/HPO User Reference Manual.

On-demand scripts
A user may have many scripts active concurrently. For instance, a user's logon profile may cause automatic initiation of several application sessions, each of which may invoke a logon script.

A script may be run at any time during the life of a session by returning to MULTSESS/HPO mode and using the RUN command. Care should of course be exercised to ensure that the current application screen image is the one that the script is expecting.

A logoff script
A logoff script may be automatically invoked using MULTSESS/HPO exit point TPEXIT02, call code 2C and 2D. These exit points are driven whenever a TERMSESS or TERMCOND command is issued to terminate a session without going through normal application logoff procedures. The exit routine may discard the TERMSESS/TERMCOND command and replace it with a RUN command to invoke an appropriate logoff script to elegantly close down the session. Use of MULTSESS exits is described in *Chapter 5 - User exits* in the MULTSESS/HPO Customization Reference Manual.

Using variables in scripts

Variable data

Variable data may be typed on behalf of a user by using the VAR script statement. The variable named on the statement will be inserted at the current cursor location. After execution of the VAR statement, the cursor will be placed at the first position following the inserted data. This feature is very useful when coding logon scripts for menu driven applications. A single general purpose script may handle many routes through the menu layers based on the contents of data variables.

Variable names may be up to 255 characters long, but cannot start with a quote (C"'",.X'7D').

Example

```
JUMPIF *,2 'ENTER USERID AND PASSWORD'
   END 4
VAR    &SYSUSER
DATA   ' '
VAR    &SYSPASS
ENTER
JUMPIF 1,15 'CICS APPLICATION MENU'
   END 8
VAR   'APPLNUM'
ENTER
JUMPIF 1.15 'PAYROLL TRANSACTION MENU'
   END 12
VAR   '&TRANID'
ENTER
END
```

Each MULTSESS user may define up to 20 variables using the ATPVAR command (or by supplying the appropriate commands in a MULTSESS profile member or a user exit routine). When the user invokes a script, either directly using a RUN command or indirectly because the application has a logon script defined in the MULTSESS ADT, the current value of all user defined variables is passed to ATP in addition to the three MULTSESS defined variables &SYSUSER, &SYSPASS and &SYSTERM.

The three MULTSESS variables representing userid, password and real terminal luname are always passed to ATP as each script starts. User defined variables will be passed up to the maximum of 20 allowed by MULTSESS, or the value specified for the MAXIMUM-VARIABLES startup parameter. Note that the value specified for this option includes the three standard variables. To ensure that all possible variables are always passed specify MAXIMUM-VARIABLES=23. Specifying a smaller value will conserve ATP storage and reduce system overhead in searching ATP's internal variable chain.

System operator interface

If the startup option STOP/MODIFY=YES is specified, the system operator may monitor and control ATP/HPO using the MVS STOP (P) and MODIFY (F) commands.

Permissible commands
Permissible commands are described under the functions of the MULTSESS/HPO ATPCMD command. (An exception is the SHUTDOWN command which is not permissible because it is affected by a P ATP command). The ATPCMD command is described in *Chapter 3 - MULTSESS/HPO ATP commands*.

For example, assuming the ATP/HPO started task name is ATP.

```
F ATP,DISPLAY *
F ATP,RESET script library-id
F ATP,TRACE ALL
F ATP,ABEND userid
P ATP
```

Writing your own scripts

The following guidelines should be borne in mind when writing your own scripts:

1. When a script is invoked as part of session initiation, the first screen seen by ATP/HPO is a blank screen. The first script statement is executed at this point. If the application normally sends a good morning message, or some other unique application identifier, the first script statement should cause a wait until this message arrives.

   ATP script initiation may on occasion be slower than the first application response, so that the first screen image seen by ATP/HPO already has the good morning message in it. Therefore a conditional wait, using the UNTIL statement is recommended.

2. For scripts designed for use within a session, rather than at session initiation, the first 'seen' by ATP/HPO is the current application screen image. It is good practice to code a test at the beginning of the script to ensure that the script has been invoked at the expected point in the application, by examining some fixed data on the screen, for instance:

   ```
   JUMPIF 1,27 'ISPF/PDF PRIMARY OPTION MENU'
   END 16
   ```

3. Script processing will be suspended whenever a script statement, for example the <Enter> key or PA key, causes data to be sent inbound to the application, or when a WAIT statement is encountered.

4. On an SNA session, script processing is resumed when an end-of-chain, OIC or LIC RU, is received with either end-bracket or change-direction.

5. On a non-SNA session, script processing is resumed when the application next sends unlock-keyboard.

6. A script finishes when a run-time error occurs or an END statement is encountered. If the last action of a script is to enter data on behalf of the user, the screen image into which the data was entered will be displayed at the user's terminal until the application sends a reply. If this screen is not to be seen by the user, a WAIT statement should be coded immediately before the END statement. This will cause the script to remain in control of the session until the application sends a reply, ensuring that the end-user sees only the result screen.

Interpreting ATP/HPO statistics

ATP/HPO can maintain connections to, and process script requests from, multiple MULTSESS/HPO address spaces.

Whenever a connection to a MULTSESS/HPO is terminated because MULTSESS/HPO or ATP/HPO has been closed down, ATP/HPO produces statistics on its log file. The following guidelines will prove useful in interpreting these statistics.

1. All script load activity is handled asynchronously by I/O subtasks within ATP/HPO. For each subtask allocated, the number of script load operations, both successful and unsuccessful, are reported. If the failure rate is high, it would indicate a possible error in the I/O subsystem, or possibly a virtual storage shortage.
2. For each script that started to load, information concerning the script is given. If the number of times a particular script is loaded is high, this may indicate a virtual storage shortage, or that the script in question should be made permanently resident, using the PERMANENT-SCRIPT startup option.
3. If the number of script completions is much less than the number of initiations, this may indicate a large number of failures. Possible causes may be an error in the script logic, or the occurrence of some nonstandard condition that the script was not designed to handle, eg password expiry.
4. General figures are reported for module storage used, script storage used and the number of script loads that were delayed by queuing for a free I/O subtask.

   If the script storage used is equal to the value specified for the SCRIPT-STORAGE startup parameter, this could indicate that more storage needs to be allocated for holding in-storage scripts in order to prevent script requests being delayed pending I/O load operations.

   If the value reported for scripts queuing for I/O is nonzero, script requests were delayed because no free subtask was available to load the requested script into storage. Increasing script storage may ease the problem by allowing more scripts to be held in-storage. Increasing the MAXIMUM-SUBTASKS startup option will allow more I/O requests to be processed concurrently.

Integrity and performance

Address space
ATP runs in its own address space. A single ATP/HPO address space may provide script services for any number of applications running in the same or different VTAM domains.

The ATP address space can and should be run at a different dispatching priority from MULTSESS and other applications, ensuring that ATP/HPO users do not receive preferential service over normal users and that response times to MULTSESS/HPO and other on-line applications are not affected. ATP/HPO can even be run on a different CPU. Only a VTAM connection is required.

Address space separation
Address space/virtual machine separation provides increased integrity and isolates other users from loops and other problems in user written scripts. It also ensures that ATP or CPU requirements do not affect other users.

ATP/HPO commands
Commands are available to:

- display system data
- reload any modified scripts dynamically
- set trace options, etc.

For ease of use these commands can be issued from the MVS system operator console or by an authorized MULTSESS/HPO user.

This page intentionally left blank

Chapter 3 - MULTSESS/HPO ATP commands

This section describes the MULTSESS/HPO commands that are specific to ATP/HPO. All commands are grouped alphabetically.

Each MULTSESS/HPO command and subcommand is assigned a command class. A command or subcommand may only be invoked by a user of equal or higher authority level.

Within each command the subfunctions are grouped by class. The default command class is indicated within each command description.

Your administrator may have changed the class of some of the commands as part of the installation process.

Command authority
A user is assigned a command authority by the system administrator in the MULTSESS/HPO directory of users.

Minimum abbreviation
For each of the commands the minimum abbreviation is indicated.

Online help
Online HELP information is available for the commands by typing:

```
HELP commandname
```

For example:

```
HELP ATPCMD
```

Entering the HELP command without a *commandname* will list all the commands for which the user is authorized.

See the HELP command for further details.

ATPCMD

Description	Allows a MULTSESS/HPO user to issue commands to the ATP address space.
Command class	**B**
Minimum abbreviation	ATPCMD
Format	**ATPCMD function**

Available functions

ABEND	abend ATP/HPO
DISPLAY	display ATP/HPO status information
RESET	mark in-store ATP/HPO script for deletion
SHUTDOWN	shutdown ATP/HPO
TRACE	start or stop ATP/HPO tracing

The functions are individually described on the following pages.

ATPCMD ABEND

Description

ATP/HPO will be abended. A dump will be provided if a SYSUDUMP card is present in the ATP/HPO startup JCL. If a userid is passed to ATP/HPO with the ABEND request, control blocks, related to ATP/HPO requests currently in process on behalf of the named user, will be formatted and printed on the ATP/HPO log file.

Command class

B

Minimum abbreviation

ATPCMD ABEND

Format

ATPCMD ABEND
ATPCMD ABEND *userid*

where:

userid	is the userid to be passed to ATP/HPO with the ABEND request.

ATPCMD DISPLAY

Description

ATP/HPO will return the following status information about the ATP/HPO address space:

- the amount of storage used by ATP/HPO modules
- the amount of storage used for scripts
- the amount of dynamic working storage in use by ATP/HPO
- the number of times the ATP/HPO maximum subtasks was exceeded, resulting in user requests being queued awaiting use of a subtask processor.

Command class

B

Minimum abbreviation

ATPCMD DISPLAY

Format

ATPCMD DISPLAY

ATPCMD RESET

Description	ATP/HPO will mark the in-storage copy of the named script from the indicated library for deletion when its use count reaches zero. The next request for this script will cause a fresh copy to be loaded from the script library. The default is the PUBLIC library.
Command class	**B**
Minimum abbreviation	ATPCMD RESET

Format

ATPCMD RESET *script*

ATPCMD RESET *script libraryid*

where:

script	is the script to be deleted.
libraryid	is the script library to load from.

ATPCMD SHUTDOWN

Description ATP/HPO will perform an orderly shutdown and produce run-time statistics.

Command class **B**

Minimum abbreviation ATPCMD SHUTDOWN

Format **ATPCMD SHUTDOWN**

ATPCMD TRACE

Description	This command activates or terminates tracing within the ATP/HPO address space. Records that trace ATP/HPO internal activity and the dataflow between MULTSESS and ATP/HPO will be written to the file pointed to by the ATPLOG DD statement in the JCL used to initiate ATP/HPO.
Command class	**B**
Minimum abbreviation	ATPCMD TRACE

Format

```
ATPCMD   TRACE   ALL
ATPCMD   TRACE   OFF
ATPCMD   TRACE   ON
```

where:

ON	is the start of minimal ATP/HPO tracing.
ALL	is the start of full ATP/HPO tracing.
OFF	stops ATP/HPO tracing.

ATPDIR

Description	Performs an on-line update of the ATP/HPO control table. The ATP/HPO table is an optional security file to control access to the scripts held in the public or private script libraries.
Command class	**C**
Minimum abbreviation	ATP
Format	**ATPDIR** ***member***

where:

member	must be the name of a valid member residing in the dataset referred to by the ATPDIR DD statement in the JCL used to initialize MULTSESS/HPO. The member will be verified for syntax before any update is attempted.

ATPVAR

Description

Defines, deletes, amends or displays the value of ATP/HPO user variables. User variables may be included in an ATP/HPO script to vary the run time operation of a script. For example a single general purpose script may initiate a session to a menu driven application, with the actual menu selections being driven by variables. Up to twenty variables may be set by each user.

Command class

G

Minimum abbreviation

ATPV

Format

ATPVAR SET *&var value*
ATPVAR DELETE *&var*
ATPVAR DELETE *
ATPVAR QUERY *&var*
ATPVAR QUERY *

where:

&var	is the name of an ATP/HPO variable.
value	is the value to be assigned to the variable.
*****	indicates all ATP/HPO variables.

Note

Any of the system variables, for example, &SYSTIME, &SYSDATE, may be used as part of *value* and will be resolved before being passed to the ATP/HPO.

RUN

Description The RUN command is used to initiate an ATP/HPO script on an already established session**.**

Command class **G**

Minimum abbreviation RUN

Format

RUN ***script*** ***alias***
RUN ***script*** ***alias*** ***trace-level***
RUN ***script*** ***alias*** **RECON**
RUN ***script*** ***alias*** **RECON** ***trace-level***

where:

script	is the name of the script to be invoked.
alias	is the alias name of the session on which to run the script.
RECON	indicates that you wish to be immediately switched into the named session. If omitted, you will remain in MULTSESS mode. The script will continue execution in the background, while you are free to continue with other work. You may reconnect to the application at a later time to view the results of the script.
trace-leve*l*	during script development, you may wish to trace the execution of your script. This option is available irrespective of the status of the MULTSESS/HPO and ATP/HPO application trace options. Use of this operand will result in trace records being written to the ATP/HPO LOG file. Three trace levels are available as shown below. The values shown are cumulative. To select more than one option, specify the sum of their individual values:

1. All data received by ATP/HPO from an application will be printed in HEX and EBCDIC.
2. All data sent by ATP/HPO on behalf of the terminal will be printed in HEX and EBCDIC.
4. The statement number of the script currently being executed and the total number of statements executed so far will be printed. The statement number is relative to the start of the script (including comment cards).

TERMRUN

Description	The TERMRUN command is used to unconditionally terminate a script running on an application session.
Command class	**G**
Minimum abbreviation	TERMR
Format	**TERMRUN** ***alias***

where:

alias	is the alias of the session on which the script is running.

Note:

At termination time ATP/HPO will return the relative statement number that was to be executed next and the screen image produced so far. The application session will remain active. You may reconnect to the session to view the screen image produced by ATP/HPO and continue application processing from that point.

The information resulting from this command will be useful if an ATP/HPO script is suspected of hanging.

This page intentionally left blank

Chapter 4 - ATP/HPO language statements

Writing scripts

This section describes all language statements that are available when writing scripts for ATP/HPO.

Syntax rules

1. Statements are free format and can start in any column.
2. Operands must be separated from commands and each other by one or more blanks.
3. Commands may be specified in uppercase or lowercase.
4. Maximum statement length (i.e. the maximum LRECL of a script dataset) is 1024 bytes.
5. Continuation statements are not supported.

Other rules

1. Every script must contain at least one END statement.
2. The following statements cannot be the next-to-last statement:

   IF

   JUMPIF

   WHILE

   UNTIL
3. A build error will occur if a script contains non-executable statements. For example, an unconditional GOTO must be followed by a LABEL statement.
4. A WHILE or UNTIL statement must be followed by one of the following:

   WAIT

   LOCK

   PFKEY

   PAKEY

   ENTER

   CLEAR
5. The last statement in a script must be either an END or a GOTO statement.
6. Comment statements are ignored when assessing all language rules. For example, a JUMPIF statement may be immediately followed by a comment, with the statement to be 'jumped' following the comment.

ATP language statements

statement	parameter	description
*	*text*	comment statement
CLEAR		simulates CLEAR key
CURSOR	*row,col*	explicit position cursor
DATA	*'text'*	simulates user typing
DOWN-CURSOR	*number*	moves cursor down screen
END	*number*	returns to caller
ENTER		simulates ENTER key
ERASE-EOF		simulates ERASE-EOF key
GOTO	*label*	branches to script label
IF	*data, data*	tests screen data
JUMPIF	*data, data*	tests data and branch
LABEL	*labelname*	'GOTO' point
LEFT-CURSOR	*number*	moves cursor left
LOCK		awaits next outbound unlock keyboard
LOG	*'datastring'*	outputs a message to the ATP log
MSG	*n 'datastring'*	passes data to MULTSESS
NON-SDLC-GOTO	*label*	branches to script label - only if non-SDLC session
NON-SDLC-WAIT		awaits next outbound send - only if non-SDLC session
PAKEY	*number*	simulates PA key
PFKEY	*number*	simulates PF key
RIGHT-CURSOR	*number*	moves cursor right
SDLC-GOTO	*label*	branches to script label - only if SDLC session
SDLC-WAIT		awaits next EB or CD - only if SDLC session
UNTIL	*data, data*	tests screen data
UP-CURSOR	*number*	moves cursor up screen
VAR	*varname*	inserts data from variable
WAIT		awaits outbound send, EB or CD - all session types
WHILE	*data, data*	test screen data

*

Description

The asterisk (*) allows comments to be included in scripts as in-line documentation.

Comment statements are stripped out as scripts are loaded into storage and so do not consume script area storage and may be ignored when assessing script structure rules. They count, however, when script statement numbers are reported.

Minimum abbreviation

*

Format

*   *text*

where:

text

can be any characters.

CLEAR

Description

This statement simulates the keyboard <Clear> key.

The <Clear> key aid byte to be sent to the application and the in-core screen buffer are cleared to nulls.

The reply mode is set to the default (field).

If the session is capable of primary and alternate screen sizes, primary mode will be entered.

The cursor is moved to buffer offset 0.

If the session is capable of partitioning, implicit partition mode will be entered.

Execution of the script is suspended until the keyboard is unlocked again by the application.

Minimum abbreviation

CLEAR

Format

CLEAR

CURSOR

Description

This statement causes the CURSOR to be positioned at an explicit location in the screen buffer.

Minimum abbreviation

CURSOR

Format

CURSOR *row,col*

where:

row

is the row where the cursor is to be placed.

col

is the column where the cursor is to be placed.

Example

```
CURSOR 16,19
```

logically places the cursor at row 16, column 19.

Possible errors

Cursor position specified is not within the current screen size.

Script terminates with error code 2.

DATA

Description

This statement simulates the user typing data. The specified datastring is inserted into the screen buffer at the current cursor location.

Minimum abbreviation

DATA

Format

DATA '*datastring*'

where:

datastring

is a data string of less than 256 characters. It must be enclosed within quotes, and may contain any character. If a quote is required as part of the string, two quotes must be specified. The enclosing quotes are removed when forming the resultant value. The data can be any character equal to or greater than X'40' or X'00'.

Example

```
DATA ‘this is a data string’
DATA ‘the quote symbol “ can be used’
```

Note

Data will be inserted in a similar manner to a real keyboard. After each byte is inserted, the cursor will be moved one position to the right. If an attribute is encountered, this will be jumped and autoskip will take effect. At the right hand side of the screen, further input will be entered on the row below. At the bottom of the screen, further input will be entered on the top row.

Possible errors

Attempting to insert data into a protected field.

Script terminates with error code 4.

DOWN-CURSOR

Description

This statement moves the cursor down the screen buffer the number of rows specified.

Minimum abbreviation

DOWN-CURSOR

Format

DOWN-CURSOR *number*

where:

number

specifies the number of rows to be moved.

Specify a numeric value less than 256.

The default is 1.

If 0 is specified, then 1 is assumed.

Example

```
DOWN-CURSOR 4
```

moves the cursor down 4 rows.

Note

Cursor movement proceeds as a series of single row moves.

If the cursor is positioned on the bottom row, a further move will position it on the top row.

END

Description

This statement terminates script processing.

Minimum abbreviation

END

Format

END *number*

where:

number

is an optional user return code to be passed back to the caller. Specify a numeric value less than 256.

The default = 0.

This parameter is particularly useful for reporting unexpected conditions arising during script execution.

Example

```
*
* Check for executed prompt on line 1.
* Error code 16 if not found.
*
JUMPIF 1,* 'ENTER USERID'
 END 16
*
* Type userid and end normally
*
VAR &SYSUSER
ENTER
WAIT
END
```

Note

Script terminates with error code 0.

ENTER

Description

This statement simulates the keyboard <Enter> key. Modified buffer contents are sent to the application with the ENTER aid byte and the current cursor address.

Script processing is suspended until the application 'unlocks' the keyboard.

For an SDLC session, this will be when EB or CD (end-bracket or change-direction) is received from the application.

For a non-SDLC session, this will be the next outbound send with the 'unlock keyboard' bit set.

Minimum abbreviation

ENTER

Format

ENTER

ERASE-EOF

Description

This statement simulates the keyboard <Erase-EOF> key. The current field is erased from the current cursor location to the next attribute character. The cursor location is unaltered.

Minimum abbreviation

ERASE-EOF

Format

ERASE-EOF

Possible errors

Current cursor location in protected field.

Script terminates with error code 2.

GOTO

Description

This statement is used to branch to the specified LABEL statement within the script.

Minimum abbreviation

GOTO *label*

Format

GOTO *label*

where:

label

is the label name specified on a LABEL statement within the same script. The names specified on the GOTO and LABEL must match exactly, that is same length, uppercase and lowercase letters, for the label to be found.

Example

```
GOTO ENTER-the-USERID
LABEL ENTER-the-USERID
```

Notes

The label to be branched to may occur either before or after the GOTO statement, that is forward and backward branching is supported.

If the label name occurs on more than one LABEL statement in the script, the GOTO will always branch to the first occurrence of the label name.

Possible errors

GOTO label cannot be found in the script.

Script terminates with error code 5.

IF

Description

The data represented by the first operand is tested against the data represented by the second operand. If the comparison tests TRUE, the statement following the IF is executed. If the comparison tests FALSE, the statement following the IF is skipped.

Minimum abbreviation

IF

Format

```
IF  (row,col       )    ('datastring' )
    ('datastring'  )    (variable     )
    (variable      )
```

where:

row

is the row on which *datastring* must start for the test to be satisfied.

col

is the column in which *datastring* must start for the test to be satisfied. If 0 or * is specified, the data may start in any column. Specifying *,* indicates that the data may be present anywhere on the screen.

datastring

is data to be tested, up to 255 bytes in length, and may contain characters equal to or greater than X'40'. It must be enclosed within quotes. If a quote is required as part of the data, two quotes must be specified. Enclosing quotes are removed before making the comparison.

variable

is the name of a variable whose contents are to be tested.

continued ....

Example

```
*
* Test for production or test IMS system
* (Take GOTO branch if the IF test is true)

IF 2,20 'PRODUCTION IMS SYSTEM'
   GOTO LOGON-PROD-IMS
*
*Code for test system logon
*
LABEL LOGON-TEST-IMS
WAIT
END
*
* Code for production system logon
*
LABEL LODON-PROD-IMS
WAIT
END
```

Notes

The data entered is compared exactly as specified, i.e. uppercase and lowercase letters matching exactly with the following exceptions:

if the NULL=BLANK startup option is specified, nulls (X'00') in the screen buffer are treated as blanks when comparing with the specified *datastring*.

if the ATTRIBUTE=BLANK startup option is specified, attribute bytes in the screen buffer are treated as blanks when comparing with the specified *datastring*.

The IF statement implements the logical NOT of the JUMPIF statement, that is the JUMPIF test is identical to the IF test, but the TRUE and FALSE meanings are reversed.

Possible errors

Cursor position specified is not within the current screen buffer.

Script terminates with error code 2.

JUMPIF

Description

The data represented by the first operand is tested against the data represented by the second operand. If the comparison tests TRUE, the statement after the JUMPIF is skipped. If the comparison tests FALSE, the statement after the JUMPIF is executed.

Minimum abbreviation

JUMPIF

Format

```
JUMPIF  (row,col       )    ('datastring' )
        ('datastring'  )    (variable     )
        (variable      )
```

where:

row
: is the row on which *datastring* must start for the test to be satisfied. If 0 or * is specified, the data may start on any row.

col
: is the column in which *datastring* must start for the test to be satisfied. If 0 or * is specified, the data may start in any column. Specifying *,* indicates that the data may be present anywhere on the screen.

datastring
: is data to be tested, up to 255 bytes in length, and may contain any characters equal to or greater than X'40'. It must be enclosed within quotes. If a quote is required as part of the data, two quotes must be specified. The enclosing quotes are removed before making the comparison.

variable
: is the name of a variable whose contents are to be tested.

continued ....

JUMPIF - continued

Example

```
* Test for production or test IMS system
* (Jump over the GOTO if JUMPIF test is true)
*
JUMPIF 2,20 ‘PRODUCTION IMS SYSTEM’
   GOTO LOGON-TEST-IMS
*
* Code for production system logon
*
LABEL LOGON-PROD-IMS
WAI
END
*
* Code for test system logon
*
LABEL LOGON-TEST-IMS
WAIT
END
```

Notes

The data entered is compared exactly as specified, i.e. uppercase and lowercase letters must match exactly with the following exceptions:

if the NULL=BLANK startup option is specified, nulls (X’00’) in the screen buffer are treated as blanks when comparing with the specified *datastring*.

if the ATTRIBUTE=BLANK startup option is specified, attribute bytes in the screen buffer are treated as blanks when comparing with the specified *datastring*.

The JUMPIF statement implements the ‘logical NOT’ of the IF statement, that is the IF test is identical to the JUMPIF test, but the TRUE and FALSE meanings are reversed.

Possible errors

Cursor position specified is not within the current screen buffer.

Script terminates with error code 2.

LABEL

Description

This statement is used to indicate a position that can be branched to from a GOTO statement.

Minimum abbreviation

LABEL

Format

LABEL *label-name*

where:

label-name

is the name of the label, less than 256 characters with no imbedded blanks.

The label must exactly match the name specified on the GOTO for it to be found, i.e. matching uppercase and lowercase letters.

Example

```
GOTO THIS-is-a-BRANCH-point
.....................
.....................
.....................
LABEL THIS-is-a-BRANCH-point
```

Note

If a LABEL is defined more than once, the first occurrence will be found by a GOTO.

LEFT-CURSOR

Description

This statement moves the cursor left along the screen buffer the number of columns specified.

Minimum abbreviation

LEFT-CURSOR

Format

LEFT-CURSOR *number*

where:

number

specifies the number of screen locations to be moved.

Specify a numeric value less than 256.

The default is 1.

If 0 is specified, then 1 is assumed.

Example

```
LEFT-CURSOR 16
```

moves the cursor 16 positions to the left.

Note

Cursor movement proceeds as a series of one position moves. For each single position move, the following rules apply:

- if the cursor is anywhere in the screen buffer other than column 1, the cursor will move to the left
- if the cursor is in row 1, column 1, the cursor is moved to the rightmost column of the bottom row
- if the cursor is in column 1 of any row other than row 1, it will move to the rightmost column of the row above

LOCK

Description

This statement causes the script to wait until the next outbound send that unlocks the keyboard has been received.

For SDLC sessions, this statement is no different to the WAIT statement.

However, for non-SDLC sessions a WAIT statement will wait for the next outbound send, whereas a LOCK will wait for the next outbound send that unlocks the keyboard.

Minimum abbreviation

LOCK

Format

LOCK

LOG

Description

This statement is used to output a message on the ATP log.

Minimum abbreviation

LOG

Format

LOG '*datastring*'

where:

datastring

is the data to be output on the ATP log up to 114 bytes in length.

Example

```
LOG 'call I executed'
......
......
......
......
```

MSG

Description

This statement is used to pass data to MULTSESS/HPO. The processing performed by MULTSESS/ HPO depends on the level entered for the command (which can be 0 to 5 as described below), and whether the function specified by this level is supported by MULTSESS/HPO at the time of invocation.

Minimum abbreviation

MSG

Format

MSG *n* '*datastring*'

where:

n

indicates what MULTSESS/HPO will do with the data. This is a number between 0 and 5 as follows:

0	-	write data to the MULTSESS log
1	-	pass data to user (display on the screen)
2	-	write data out as a WTO
3	-	execute data as a MULTSESS command
4	-	(reserved for future use)
5	-	execute data as a MULTSESS command after script terminates

datastring

is the data to be passed to MULTSESS/HPO for further processing and can be up to 255 bytes in length.

Example

```
MSG 0 'SCRIPT FOR USER TSG1 COMPLETED'
```

NON-SDLC-GOTO

Description

This statement is used to branch to the specified LABEL statement within the script, only if the script is running on a non-SDLC session.

If a script containing a NON-SDLC-GOTO statement is run on an SDLC session, it will be ignored. This enables a single script to be shared for use on behalf of users of SDLC and non-SDLC terminals when in session with applications which are sensitive to session protocol.

In all other respects, this statement is the same as the GOTO statement.

Minimum abbreviation

NON-SDLC-GOTO *label*

Format

NON-SDLC-GOTO *label*

where:

label

is the label name specified on a LABEL statement within the same script. The names specified on the GOTO and LABEL must match exactly, i.e. same length, uppercase and lowercase letters, for the label to be found.

NON-SDLC-WAIT

Description

After data has been sent inbound on a non-SDLC session, for example by an ENTER or PFKEY statement, the ATP/HPO script is suspended until the application sends data with the 'unlock keyboard' bit set.

The NON-SDLC-WAIT statement causes the script, if it is running on a non-SDLC session, to be suspended even though the keyboard has been unlocked by the application.

The NON-SDLC-WAIT will last until the next outbound send from the application.

If a script containing a NON-SDLC-WAIT statement is run on an SDLC session, it will be ignored. This enables a single script to be shared for use on behalf of users of SDLC and non-SDLC terminals when in session with applications which are sensitive to session protocol.

Minimum abbreviation

NON-SDLC-WAIT

Format

NON-SDLC-WAIT

Related statements

SDLC-WAIT
WAIT
LOCK

PAKEY

Description

This statement simulates the Program Attention (PA) keys on the terminal. The appropriate PA key aid byte is sent to the application.

Note that, as with a real terminal, the PAKEY statement causes a short read. Modified data is NOT sent to the application as part of the datastream.

Script processing is suspended until the application 'unlocks' the keyboard. For an SDLC session, this will be when EB or CD (end-bracket or change-direction) is received from the application. For a non-SDLC session, this will be the next outbound send with the unlock keyboard bit set.

Minimum abbreviation

PAKEY

Format

PAKEY *number*

where:

number

is the number of the PA key to be simulated, in the range 1 to 3.

Example

```
PAKEY 2
```

sends <PAK2> to the application.

PFKEY

Description

This statement simulates the Program Function (PF) keys on the terminal. Modified buffer contents are sent to the application with the appropriate PF key aid byte and the current cursor address.

Script processing is suspended until the application unlocks the keyboard. For an SDLC session, this will be when EB or CD (end-bracket or change-direction) is received from the application. For a non-SDLC session, this will be the next outbound send with the unlock keyboard bit set.

Minimum abbreviation

PFKEY

Format

PFKEY *number*

where:

number

is the number of the PF key to be simulated, in the range 1 to 24.

Example

```
PFKEY 16
```

sends <PFK16> to the application.

RIGHT-CURSOR

Description

This statement moves the cursor to the right along the screen buffer by the number of columns specified.

Minimum abbreviation

RIGHT-CURSOR

Format

RIGHT-CURSOR *number*

where:

number

specifies the number of screen locations to be moved. Specify a numeric value less than 256.

The default is 1

If 0 is specified, then 1 is assumed.

Example

```
RIGHT-CURSOR 14
```

moves cursor 14 positions to the right.

Note

Cursor movement proceeds as a series of one position moves. For each single position move, the following rules apply:

- if the cursor is anywhere in the screen buffer other than the rightmost column, the cursor is moved to the right
- if the cursor is in the rightmost column of the bottom row, it is moved to row 1 column 1
- if the cursor is in the rightmost column of any row other than the bottom row, it is moved to column 1 of the row below.

SDLC-GOTO

Description

This statement is used to branch to the specified LABEL statement within the script, only if the script is running on an SDLC session.

If a script containing an SDLC-GOTO statement is run on a non-SDLC session, it will be ignored. This enables a single script to be shared for use on behalf of users of SDLC and non-SDLC terminals when in session with applications which are sensitive to session protocol.

In all other respects, this statement is the same as the GOTO statement.

Minimum abbreviation

SDLC-GOTO

Format

SDLC-GOTO *label*

where:

label

is the name of the label, less than 256 characters with no imbedded blanks.

The label must exactly match the name specified on the GOTO for it to be found, i.e. matching uppercase and lowercase letters.

SDLC-WAIT

Description

After data has been sent inbound on an SDLC session, for example by an ENTER or PFKEY statement, the ATP/HPO script is suspended until the keyboard is unlocked by a change-direction (CD) or end-bracket (EB) received from the application.

The SDLC-WAIT statement causes the script, if it is running on an SDLC session, to be suspended even though the keyboard has been unlocked by the application.

The SDLC-WAIT will last until the next EB or CD is received from the application.

If a script containing an SDLC-WAIT statement is run on a non-SDLC session, it will be ignored. This enables a single script to be shared for use on behalf of users of SDLC and non-SDLC terminals when in session with applications which are sensitive to session protocol.

Minimum abbreviation

TERMC

Format

SDLC-WAIT

Related statements

NON-SDLC-WAIT
WAIT
LOCK

UNTIL

Description

This statement causes ATP/HPO to continually execute the single statement immediately following the UNTIL statement, until the data represented by the two operands match.

Minimum abbreviation

UNTIL

Format

```
UNTIL (row,col       ) ('datastring'  )
      ('datastring'  ) (variable      )
      (variable      )
```

where:

row
: is the row on which *datastring* must start for the test to be satisfied.

  If 0 or * is specified, the data may start on any row.

col
: is the column in which *datastring* must start for the test to be satisfied.

  If 0 or * is specified, the data may start in any column.

  Specifying *,* indicates that the data may be present anywhere on the screen.

datastring
: is data to be tested, up to 255 bytes in length, and may contain any characters equal to or greater than X'40'.

  It must be enclosed within quotes. If a quote is required as part of the data, two quotes must be specified. The enclosing quotes are removed before making the comparison.

variable
: is the name of a variable whose contents are to be tested.

continued ....

UNTIL - continued

Example

```
*
*          Logoff from TSO/ISPF
*Press PFK3 until out of ISPF and the TSO
*'READY' message appears (column 2 of any row)
*
UNTIL  *,2 'READY'
PFKEY 3
DATA  'LOGOFF'
ENTER
END
```

Notes

The data entered is compared exactly as specified (i.e. uppercase and lowercase letters must match exactly), with the following exceptions:

if the NULL=BLANK startup option is specified, nulls (X'00') in the screen buffer are treated as blanks when comparing with the specified *datastring*.

if the ATTRIBUTE=BLANK startup option is specified, attribute bytes in the screen buffer are treated as blanks when comparing with the specified *datastring*.

Possible errors

Cursor position specified is not within the current screen buffer.

Script terminates with error code 2.

UP-CURSOR

Description

This statement moves the cursor up the screen buffer by the number of rows specified.

Minimum abbreviation

UP-CURSOR

Format

UP-CURSOR   *number*

where:

number

specifies the number of rows to be moved.

Specify a numeric value less than 256.

The default is 1.

If 0 is specified, 1 is assumed.

Example

```
UP-CURSOR 6
```

moves the cursor up 6 rows.

Note

Cursor movement proceeds as a series of single row moves. If the cursor is positioned on the top row, a further move will position it on the bottom row.

VAR

Description

This statement will insert the current value of the variable in the screen buffer at the current cursor position.

Minimum abbreviation

VAR

Format

VAR *varname*

where:

varname

is the name of the variable.

The following variables are automatically made available for use in any ATP/HPO script:

&SYSUSER - userid,

&SYSPASS - password,

&SYSTERM -the real terminal nodename,

&SYSUSER, &SYSPASS and &SYSTERM must be typed in uppercase characters.

Note

If the specified variable is not currently defined, the statement is ignored. Data will be inserted in a similar manner to a real keyboard. After insertion of each byte, the cursor will be moved one position to the right.

If an attribute is encountered, this will be jumped and autoskip will take effect. On the right hand side of the screen, further input will be entered on the row below. At the bottom of the screen, further input will be entered on the top row.

Possible errors

Attempting to insert data into a protected field.

Script will terminate with an error code 4.

WAIT

Description

The WAIT statement is used to suspend an ATP/HPO script and can be specified with or without a parameter of *nnnn* to limit the length of the wait.

If specified without the *nnnn* parameter, the WAIT statement acts in the following way:

after data has been sent inbound to the application, for example by an ENTER or PFKEY statement, the ATP/HPO script is suspended until:

- for an SDLC session, an end-bracket (EB) or change direction (CD) is received
- for a non-SDLC session, a keyboard unlock is received.

Note: Some applications, notably TSO at logon time, can unlock the keyboard before all outbound data has been delivered to the terminal. In this case the WAIT statement causes the script to be suspended even though the keyboard has been unlocked by the application.

Therefore the WAIT will last until:

- for an SDLC session, the next EB or CD is received
- for a non-SDLC session, the next outbound send.

If specified with the *nnnn* parameter, the WAIT times out if either the keyboard is unlocked or the time specified by *nnnn* has been exceeded. This prevents a script waiting for an event that never happens.

Minimum abbreviation

WAIT

Format

WAIT [*nnnn*]

where

nnnn is the time limit for the WAIT command, in tenths of a second, up to a maximum of 9999 (999.9 seconds).

continued ....

WAIT - continued

Examples

WAIT is typically used in conjunction with a conditional test statement, such as UNTIL, IF etc. The following example illustrates the use of UNTIL/WAIT during a logon to TSO, to suspend entry of the userid until TSO is ready for it.

```
*
*  WAIT FOR PROMPT FOR USERID ON LINE 1
*
UNTIL 1,*  'ENTER USERID'
  WAIT
*
*  'TYPE' USERID AND PRESS ENTER
*
VAR  &SYSUSER
ENTER
```

Note: Use of UNTIL with the WAIT *nnnn* statement will not timeout.

The next example illustrates the use of the WAIT command using the timeout parameter.

```
IF 1,2 'Enter Password'
   GOTO enter-pass
WAIT 50
IF 1,2 'Enter Password'
   GOTO enter-pass
WAIT 50
IF 1,2 'Enter Password'
   GOTO enter-pass
MSG 1 'Script failed user intervention reqd'
GOTO all-done
VAR &SYSPASS
ENTER
LABEL all-done
END
```

If the required data is already on the screen when the script starts, the password is entered immediately.

If the words 'Enter Password' do not appear in line 1, column 2 after 10 seconds (2 x 50 tenths of a second), the script displays the message 'Script failed user intervention reqd'.

Note: The script allows the screen to be unlocked or timed out twice so it can check if the Enter Password prompt has arrived after some unexpected data. The unexpected data can therefore be ignored. This allows a similar function to the UNTIL command but with a timeout option.

To wait on a non-SDLC session for the next send with keyboard unlock, see the LOCK command.

Related statements

SDLC-WAIT
NON-SDLC-WAIT
LOCK

WHILE

Description

This statement causes ATP/HPO to continually process the single statement immediately following the WHILE statement, as long as the data represented by the two operands match.

When the condition no longer tests true, the statement after the WHILE is skipped and script execution continues with the next script statement.

Minimum abbreviation

WHILE

Format

```
WHILE (row,col       )  ('datastring'  )
      ('datastring'  )  (variable      )
      (variable      )
```

where:

row

is the row on which datastring must start for the test to be satisfied.

If 0 or * is specified, the data may start on any row.

col

is the column in which datastring must start for the test to be satisfied.

If 0 or * is specified, the data may start in any column. Specifying *,* indicates that the data may be present anywhere on the screen.

datastring

is data to be tested, up to 255 bytes in length, and may contain any characters equal to or greater than X'40'. It must be enclosed within quotes. If a quote is required as part of the data, two quotes must be specified. The enclosing quotes are removed before making the comparison.

variable

is the name of a variable whose contents are to be tested.

continued ....

Example

```
*
* If TSO is displaying the three stars (***)
* prompt in column 2 of any line, press ENTER
*
WHILE *,2 '***'
ENTER

*
* Prompt (***) cleared so continue
*
DATA 'ISPF 3.7'
ENTER
WAIT
END
```

Note

The data entered is compared exactly as specified, i.e. uppercase and lowercase letters must match, with the following exceptions:

if the NULL=BLANK startup option is specified, nulls (X'00') in the screen buffer are treated as blanks when comparing with the specified *datastring*.

if the ATTRIBUTE=BLANK startup option is specified, attribute bytes in the screen buffer are treated as blanks when comparing with the specified *datastring*.

Possible errors

Cursor position specified is not within the current screen buffer.

Script terminates with error code 2.

Restricted keyboard functions in version 1.4

The following keyboard functions are not supported:

forward tab
backward tab
home
down tab
delete
insert

The command

ERASE ALL UNPROTECTED

is not supported.

If the terminal supports partitioning, only single partitions with an id of 1 are supported.

Chapter 5 - ATP/HPO messages

Error reporting

Errors occurring within ATP/HPO itself, disruptions and errors on the primary session between ATP/HPO and the MULTSESS/HPO address space (for which ATP/HPO is providing scripting services), or errors detected within scripts active on individual application sessions, may be reported in one or more of the following ways:

1. Messages to the system operator console, written using SVC 35 (WTO) with route codes 2 and 11. The messages and their meanings are described in *Messages to the system operator* on page 5.2.
2. If ATP/HPO encounters a critical error condition from which it is unable to recover, it will abnormally terminate with an abend code 306.
3. ATP/HPO writes a log of events to the dataset, or SYSOUT file, pointed to by the ATP/HPO LOG DD statement.

   Messages include routine, non-error, events, error conditions and shutdown statistics, and are described in *ATP/HPO log messages* on page 5.4.

Error codes
ATP/HPO is a slave application providing script services to other calling applications.

Errors occurring either on the session between ATP/HPO and its caller, or as a failure in a script request running on behalf of a user of the calling application, will be reported back to the caller as an error code.

The caller will normally interpret the failure code and use it to display or log a message of its own which describes the error. The types of error which can occur are:

1. The requested script cannot be started, for instance the script cannot be found or contains a syntax error. These codes are described in *Script load error codes* on page 5.10.
2. The script abnormally terminates, for instance data is entered into a protected field. These error codes are described in *Script termination codes* on page 5.12.
3. ATP/HPO detects a protocol error on the session between itself and a calling application and terminates the session. No further script requests from the caller can be processed. These codes should never occur in normal use but are described for completeness in *ATP/HPO termination error codes* on page 5.13.

Messages to the system operator

message	description
ATINIT	**ATPLOG DD STATEMENT NOT CODED** An attempt was made to initialize ATP/HPO without the ATPLOG DD statement.
ATINIT	**UNABLE TO LOAD MODULE ATCONS** ATP/HPO was unable to load module ATCONS at initialization time. Verify the installation procedure and ensure that the module exists in the steplib library.
ATINIT	****WARNING** YOUR ATP LICENSE EXPIRES WITHIN 14 DAYS** Your trial period expires soon. Contact your local marketing representative.
ATINIT	**A.T.P. NOT LICENSED FOR THIS CPU** This message indicates that the trial/licence period for this copy of ATP/HPO has expired. This message is issued 5 times and is non-rollable. If this message occurs during a trial period, contact your local support office for an extension. If this message occurs outside a trial period, contact your local support office to find out why the licence has expired.
ATINIT	**WAITING FOR ACB ACTIVATION - COMMAND IGNORED** ATP/HPO is waiting for activation in its main ACB. The only MVS console command valid at this time is a STOP (P) command.
ATPOPER	**INVALID COMMAND** A Modify (F) command has been entered for ATP/HPO. The command you have issued is not in the ATP command list. Correct your command syntax and reissue the command.
ATPOPER	**ABEND - IN PROGRESS** 'F ATP,ABEND' has been accepted. ATP/HPO is in the process of shutting down abnormally.
ATPOPER	**ABEND - INVALID PCB NAME** 'F ATP,ABEND userid' was entered, but ATP/HPO was unable to locate a PCB for the specified userid. The command is ignored.
ATPOPER	**TRACE - ON** 'F ATP,TRACE ON' has been accepted. Trace data will be written to the ATPLOG file.
ATPOPER	**TRACE - OFF** 'F ATP,TRACE OFF' has been accepted. ATP/HPO tracing has been halted.
ATPOPER	**TRACE - INVALID OPTION** 'F ATP,TRACE' was entered but the option you have specified is invalid.
ATPOPER	**DISPLAY - INVALID OPTION** 'F ATP,DISPLAY' was entered with an invalid option. The command is ignored.

ATPOPER	**DISPLAY - GETMAIN = *value*** In response to a DISPLAY command, ATP/HPO reports the amount of storage getmained for working storage.
ATPOPER	**DISPLAY - MODULES = *value*** In response to a DISPLAY command, ATP/HPO displays the amount of virtual storage used for ATP/HPO program modules.
ATPOPER	**DISPLAY - SCRIPTS = *value*** In response to a DISPLAY command, ATP/HPO displays the amount of virtual storage occupied by scripts.
ATPOPER	**RESET - SCRIPT NOT FOUND** 'F ATP,RESET' was entered to delete an in-storage copy of a script, but the specified script is not currently in storage.
ATPOPER	**RESET - COMPLETED** 'F ATP,RESET' has been completed. The script library combination entered on the command has been marked for deletion and will be reloaded from disk at the next reference. The script was not a permanently loaded script.
ATPOPER	**RESET - COMPLETED (PERMANENT)** 'F ATP,RESET' has been completed. The script/library combination entered on the command has been marked for deletion and will be reloaded from disk at the next reference. The script was a permanently loaded script.
ATPOPER	**RESET - INVALID SCRIPT NAME** 'F ATP,RESET' was issued but an invalid script name was specified. The command is ignored.
ATP	**RESET - INVALID LIBRARY NAME** 'F ATP,RESET' was issued but an invalid library name was specified. The command is ignored.
ATP	**ATP INITIALIZATION COMPLETE** ATP is initialized and is waiting for work.
ATP	**ATP TERMINATION COMPLETE** ATP has terminated normally as a result of a SHUTDOWN command.
ATTIMER	**ABEND** The ATPLOG writer module has abended for some reason. ATP continues to function normally but no log will be produced from now on. ATP should be restarted at the earliest opportunity.
ATP	**INITIALIZATION FAILURE** ATP/HPO initialization has failed, a separate message should explain why ATP/HPO will terminate.
ATP	**SETLOGON FAILURE** SETLOGON has failed, ATP/HPO will terminate.
ATP	**UNKNOWN ECB POSTED** A mainline has been posted, ATP/HPO will terminate.

ATP/HPO log messages

message	description
ATESTAE	***userid* - *alias* - ========= ATP ABEND ========** ATP/HPO has detected an abend in a mainline module. Other messages will follow describing the type and location of the abend.
ATESTAE	**ABEND OCCURRED AT *address* (OFFSET *offset*) IN MODULE *module* CODE S*nnn* U*nnn*** The location of the abend within an ATP/HPO mainline module is reported.
ATESTAE	**CALLED FROM *address* (OFFSET *offset*) IN MODULE *module*** The location from which the module reported in the previous message was called is reported.
ATESTAE	***userid* - *alias* - ====== ATESTAE COMPLETE =======** ATP/HPO has completed reporting a mainline code abend and is in the process of shutting down.
ATESTAI	***userid* - *alias* - ======= SUBTASK ABEND ========** ATP/HPO has detected an abend condition during execution of a subtask. Subtasks are used by ATP/HPO to load user scripts. Other messages will follow describing the location and type of abend.
ATESTAI	**SDWA** The SDWA presented to the ESTAE exit by MVS is printed in both hexadecimal and EBCDIC.
ATESTAI	**SCB** The main ATP/HPO System Control Block is printed in both hexadecimal and EBCDIC.
ATESTAI	**STB** The Subtask Control Block representing the subtask that has abended is printed in both hexadecimal and EBCDIC.
ATESTAI	**STB-ICB** The Instruction Control Block representing the script being loaded by the failing subtask is printed in both hexadecimal and EBCDIC.
ATESTAI	**STB-PCB** The Process Control Block representing each user awaiting the loading of the script is printed in both hexadecimal and EBCDIC.
ATESTAI	**ABEND OCCURRED AT *address* (OFFSET *offset*) IN MODULE *module* CODE S*nnn* U*nnn*** The location of the abend within an ATP/HPO subtask is reported.
ATESTAI	**CALLED FROM *module* (OFFSET *offset*) IN MODULE *module-name*** The location from which the module reported in the previous message was called is reported.

ATESTAI ***userid - alias*** **- === SUBTASK RECOVERY INVOKED ===**
ATP/HPO has completed reporting the subtask abend and the subtask will be re-instated to process further script load requests. ATP/HPO continues normal processing.
Note
If the script error is corrected, the script will have to be reset before ATP/HPO will load this script again.

ATINIT **FIX TABLE**
ATP/HPO is initializing and is reporting the fixes applied to the version of ATP/HPO that has been started. The fix table will follow this message and reports each possible fix as either applied (denoted by an 'X') or not yet applied (denoted by an 'O').

ATINIT **ERROR LOADING MODULE *module***
module could not be loaded during ATP/HPO initialization. ATP/HPO is terminating.

ATINIT ***** invalid card *****
ATP/HPO is reporting an invalid startup statement. The text of the message is replaced with the contents of the invalid record.

ATINIT **INVALID STARTUP PARAMETER CODED**
ATP/HPO has read a control card which is invalid. The card in error will have been printed in the log previous to this.

ATINIT **WARNING - ADDRESS SPACE RUNNING SWAPPABLE**
ATP/HPO has detected that it is not running authorized and cannot make itself non-swappable.

ATINIT **ADDRESS SPACE MARKED NON-SWAPPABLE**
ATP/HPO has detected that it is running authorized and has made itself non-swappable.

ATINIT **ERROR DURING OPEN OF ACB - ERROR CODE = X'*nn*'.**
ATP/HPO has failed to open its ACB for some reason. The error code returned by VTAM is reported in the message. For non-critical errors, ATP/HPO will wait and retry the open every 30 seconds. For details of the error code reported, refer to the description of the OPEN macro in the VTAM Programming Guide.

ATINIT **INITIALIZATION COMPLETE**
ATP/HPO has successfully initialized and is ready to accept script requests from calling applications.

ATINIT **ATP ABENDING DUE TO ABOVE ERRORS**
ATP has failed to initialize and is shutting down. Previous messages will have detailed the failures encountered.

ATINIT **DATE FORMAT INCORRECT**
The DATE= startup parameter has been incorrectly coded.

ATINIT **PRIVATE LIBRARY DSNAME PREFIX MAY NOT EXCEED 35 CHARACTERS**
The value specified for the LIBRARY-PREFIX= startup parameter contains more than 35 characters.

ATINIT **PRIVATE LIBRARY DSNAME SUFFIX MAY NOT EXCEED 35 CHARACTERS**
The LIBRARY-SUFFIX= startup parameter has been specified containing more than 35 characters.

ATINIT **COMBINED LENGTH OF PRIVATE LIBRARY PREFIX AND SUFFIX MAY NOT EXCEED 34 CHARACTERS.**
Both the LIBRARY-PREFIX= and the LIBRARY-SUFFIX= startup parameters have been specified and when combined they contain more than 34 characters.

ATINIT **PUBLIC LIBRARY NAME SPECIFIED EXCEEDS 44 CHARACTERS.**
The PUBLIC-LIBRARY= startup parameter has been specified containing more than 44 characters.

ATINIT **MAXIMUM-RUSIZE PARAMETER INVALID.**
The MAXIMUM-RUSIZE= startup parameter has been incorrectly coded.

ATINIT **INVALID TRACE OPTION SELECTED.**
A TRACE= startup parameter, other than YES, NO or ALL, has been coded.

ATINIT **MAXIMUM-SUBTASKS PARAMETER INVALID.**
The MAXIMUM-SUBTASKS= startup parameter has been incorrectly coded.

ATINIT **ACBNAME PARAMETER HAS NOT BEEN SPECIFIED.**
The ACBNAME= startup parameter has not been coded. This parameter is mandatory.

ATINIT **PERMANENT-SCRIPT PARAMETER INVALID.**
The PERMANENT-SCRIPT= startup parameter has been incorrectly coded.

ATINIT **SCRIPT-STORAGE PARAMETER INVALID.**
The SCRIPT-STORAGE= startup parameter has been incorrectly coded.

ATINIT **ESTAE ENVIRONMENT SETUP FAILURE.**
ATP/HPO has attempted to set up the ESTAE environment under which it always runs. However, this has failed for some reason. Check that the region size is sufficient.

ATINIT **MAXIMUM-GOTOS PARAMETER INVALID**
The MAXIMUM-GOTOS= startup parameter has been incorrectly coded.

ATINIT **MAXIMUM-VARIABLES PARAMETER INVALID**
The MAXIMUM-VARIABLES= startup parameter has been incorrectly coded.

ATINIT **LOG-PAGE-LENGTH PARAMETER INVALID**
The LOG-PAGE-LENGTH= startup parameter has been incorrectly coded.

ATINIT **STOP/MODIFY PARAMETER INVALID**
The STOP/MODIFY= startup parameter has been incorrectly coded.

ATINIT ***** PASSWORD SUPPRESSED *****
The PASSWORD= startup parameter has been specified but for security reasons is not listed on the log.

ATINIT ***** CODE SUPPRESSED *****
The CODE= startup parameter has been specified but for security reasons is not listed on the log.

ATP **TERMINATION COMPLETE**
ATP/HPO has been shutdown normally.

ATP **ACB HAS BEEN CLOSED**
VTAM has notified ATP/HPO that communication is no longer possible.

ATREQ09 ***userid* - *alias* - *caller* - SCRIPT *scriptname/libraryname* COMPLETE AT *statement-num* CODE *rc***
The indicated script has been terminated prematurely by the caller, for instance by a MULTSESS/HPO user issuing a TERMRUN command. Return codes and their meanings are described in *Script termination codes* on page 5.12.

ATSPROC ***userid* - *alias* - *caller* - SCRIPT *scriptname/libraryname* COMPLETE AT *statement-num* CODE *rc***
The indicated script has completed for some reason. Return codes and their meanings are described in *Script termination codes* on page 5.12.

ATSTATS ***subtask-num* - SUCCESSFUL LOADS = *nnn***
ATSTATS ***subtask-num* - FAILING LOADS = *nnn***
These messages form part of the ATP/HPO shutdown statistics. ATP/HPO reports, for each subtask allocated, the number of script loads that succeeded and the number that failed for any reason.

ATSTATS **SCRIPT '*name/library*' - PERMANENT = *nnn* -RESET=*nnn***
ATSTATS **- SCRIPT SIZE = *nnn***
ATSTATS **- TIMES LOADED = *nnn***
ATSTATS **- INITIATIONS = *nnn***
ATSTATS **- COMPLETIONS = *nnn***
These messages form part of ATP/HPO shutdown statistics. ATP/HPO reports statistical information for each script loaded.

ATSTATS **ATP/HPO MODULE STORAGE = *nnnn* BYTES**
This message forms part of the ATP/HPO shutdown statistics. ATP/HPO reports the number of bytes of storage used by the ATP/HPO modules.

ATSTATS **ATP/HPO SCRIPT STORAGE = *nnnn* BYTES**
This message forms part of the ATP/HPO shutdown statistics. ATP/HPO reports the number of bytes of storage used by ATP/HPO for storing scripts.

ATSTATS **NUMBER OF SUBTASKS QUEUED = *nnnn***
This message forms part of the ATP/HPO shutdown statistics. ATP/HPO reports the number of times a script load was needed but no subtask was free to perform the load. If the indicated value is greater than zero, you should consider increasing the value of the MAXIMUM-SUBTASKS startup parameter.

ATSTATS ***caller-acbname* - SCRIPTS STARTED = *nnnn***
ATSTATS ***caller-acbname* - SCRIPTS FINISHED = *nnnn***
This message forms part of the ATP/HPO shutdown statistics. For each caller still connected at the time ATP/HPO was shut down, ATP/HPO reports the number of scripts started and the number of scripts completed.

ATSUBT ***userid -caller* -SUBTASK INITIALIZATION COMPLETE**
This message is issued by each subtask allocated as it successfully completes initialization. It is now ready to load scripts when requested.

ATTERM ***caller-acbname* TERMINATED - ERROR CODE = *rc***
ATP/HPO has detected that a caller has failed to observe the communication rules between itself and ATP/HPO. The session is terminated and all active scripts are abandoned. For more information on the error code *rc* refer to *ATP/HPO termination error codes* on page 5.13.

ATUNBND ***caller-acbname* - SCRIPTS STARTED = *nnnn***
ATUNBND ***caller-acbname* - SCRIPTS FINISHED = *nnnn***
A calling application, for whom ATP/HPO has been providing scripting services, has terminated its session with ATP/HPO. ATP/HPO reports the number of scripts started and the number of scripts completed on behalf of the caller.

ATVLERAD **E-LERAD-RPL-UNUSABLE.**
ATP/HPO has detected that a VTAM request has failed because of an unusable RPL. The storage area that was found to be invalid is printed after this message in hexadecimal and EBCDIC.

ATVLOGON **ATTEMPTED LOGON FROM TERMINAL *luname* REJECTED**
A terminal has attempted to connect to ATP/HPO and was rejected. ATP/HPO does not support terminals.

ATVNSEXI ***caller-acbname* SESSION CLEANUP**
VTAM has notified ATP/HPO that a caller, for whom ATP/HPO was providing scripting services, is no longer available.

ATVNSEXI **UNKNOWN RU RECEIVED - REPORT THIS MESSAGE**
ATP/HPO has received a VTAM RU in its NSEXIT which it does not understand. The RPL and RU are printed after this message in hexadecimal and EBCDIC.

ATVSCIP **INVALID CONNECTION ATTEMPT**
A caller has attempted to connect with ATP/HPO but certain protocol validation parameters have been incorrectly specified. The connection attempt is rejected.

ATVSCIP ***caller-acbname* BIND RECEIVED**
ATP/HPO has been notified by VTAM that another application wishes to connect with it.

ATVSCIP **INBOUND RUSIZE EXCEEDS MAXIMUM**
A caller has attempted to start a session with ATP/HPO. However, the RUSIZE specified in the connection parameters exceeds the value specified in the ATP/HPO startup parameter MAXIMUM-RUSIZE=.

ATVSCIP **INBOUND RUSIZE < 128 BYTES**
A caller has attempted to start a session with ATP/HPO. However, the RUSIZE specified in the connection parameters for ATP/HPO-to-caller data flow is less than the minimum supported value of 128 bytes. The connection request is rejected.

ATVSCIP **UNKNOWN UNBIND RECEIVED - IGNORED**
VTAM has notified ATP/HPO that a caller wishes to terminate its session. However, ATP/HPO does not know of this caller.

ATVSCIP ***caller-acbname* UNBIND RECEIVED**

A caller has requested ATP/HPO to terminate the session between them.

ATVSCIP **OUTBOUND RUSIZE < 128 BYTES**

A caller has attempted to start a session with ATP/HPO. However, RUSIZE specified in the connection parameters for caller-to-ATP/HPO dataflow is less than the minimum supported value of 128 bytes. The connection request is rejected.

ATVTPEND **VTAM HALT COMMAND ISSUED**
ATVTPEND **VTAM HALT QUICK OR V INACT I OR F ISSUED**
ATVTPEND **VTAM HALT CANCEL ISSUED**

VTAM has driven the ATP/HPO TPEND exit to tell ATP/HPO to stop communications.

ATVXLOAD ***userid-alias-caller-* SCRIPT '*script/library*' LOAD SUCCESS AT s*tatement-count* SIZE *nnnn***

Script alias has been successfully loaded into virtual storage to satisfy a request from user *userid*. The number of (non-comment) statements and the size, in bytes, of the script are indicated.

ATVXLOAD ***userid-alias-caller* - SCRIPT '*script/library*' LOAD FAILURE AT *statement-number* CODE *rc* TYPE *tc***

A script load has failed. The relative number of the last statement loaded (excluding comments) is reported along with the error code (*rc*) and error type (*tc*). For more information on these error codes refer to *Script load error codes* on page 5.10.

ATVXRECV **INVALID REQUEST-CCB.**
ATVXRECV ***userid - alias* - INVALID REQUEST-RPL**

ATP/HPO has received invalid data from a caller. The Caller Control Block (CCB), RPL and RU are printed after these messages in hexadecimal and EBCDIC.

Note

The messages listed in this section may be issued in normal operation of ATP/HPO. In addition, numerous further messages will be written to the ATPLOG file if tracing is active. These are primarily intended for use by product support personnel in diagnosing problems and are not included here.

Script load error codes

Running scripts

When ATP/HPO is requested to run a script which is not already in storage, it is loaded from the indicated library. The script statements are read from disk, comment statements are discarded, the script syntax is checked and the statements are converted to a compressed internal format. A number of error conditions may be encountered and are reported by an appropriate error code.

Error code reporting

The code will be reported on the ATPLOG file as part of the routine reporting of script load activities. The code is also returned to the calling application so that the end user may be informed of the error.

Major code reporting

The major codes are never reported directly to the MULTSESS/HPO user. MULTSESS/HPO interprets the code and uses it to display message MS0201, indicating a script load error has occurred, followed by a further message, in the range MS0202-MS0207, to describe the error in more detail.

Syntax error reporting

The subcodes associated with error code 7, a syntax error, are reported to the end user as part of MULTSESS/HPO message MS0206 to indicate the exact nature of the error.

Error code listing

The codes which can occur, together with meaning, are listed in *Error codes* on page 5.11.

Error codes

code	description
0	Script loaded successfully.
1	Script must be loaded. This code is for internal use and is never reported to the user. A further code is returned when the load completes.
2	Script is in the process of loading. This code is for internal use and is never reported to the user. A further code is returned when the load completes.
3	An abend occurred while loading the script.
4	Script library could not be found.
5	Script could not be found in the requested library.
6	Script could not be read.
7	Script contained a syntax error. One of the following sub-codes further describes the error: 1 - an invalid script statement has been entered. 2 - a required operand is missing. 3 - an operand has unmatched quotes. 4 - an operand contains an invalid digit. 5 - spurious characters have been found after the statement. 6 - script does not contain at least one END statement. 7 - The statement is not allowed in this position. 8 - The statement was too long. 9 - an incorrect operand value has been specified.
8	No script storage was available to load the script.

Script termination codes

Completion codes
When a script terminates, one of the following completion codes is returned.

Code	Description
0	Script successful.
1	Maximum GOTO limit exceeded.
2	Invalid cursor address detected.
3	Invalid Script command encountered.
4	Data entered into protected field.
5	Goto label cannot be found.
6	Script terminated by user.
n	The user code specified on the END statement.

Error code
The error code is returned to the calling application along with the number of the last statement executed and the number of script statements executed are returned.

Code 0
Code 0 will result in message MS0218 unless the user is in dynamic panel mode, in which case no message is produced on successful script completion.

Codes 1 to 6
For codes 1 to 6, MULTSESS/HPO will interpret the code and display a message in the range MS0209 to MS0214.

User code
A user code, a value greater than 6 specified on an END statement, will be reported in message MS0219.

ATP/HPO termination error codes

Invalid dataflow termination
When ATP/HPO detects an invalid dataflow between itself and a caller the session is terminated. The reason for this termination is written to the ATP/HPO log.

The termination codes are as follows:

Code	Description
1	The VTAM RPL settings are incorrect.
2	The DRH header is invalid.
3	The DRHPCB address is invalid.
4	The DRH chaining is invalid.
5	The DRH Lutype is incorrect.
6	A readbuf has already been processed.
7	A readbuf has not yet been processed.
8	Script not yet loaded.
9	Readbuf invalid format.
10	Variable format invalid.
11	Readbuf request not allowed in chain.

Data Request Header
ATP/HPO protocol. The Data Request Header (DRH) prefixes all data passed between ATP/HPO and a calling application and enforces the data transfer used by ATP/HPO.

These types of errors indicate a serious logic error has taken place and should be immediately reported to your local support office.

This page intentionally left blank

Appendix - ATP/HPO sample script

```
*
* The following script will:
*    Log on to TSO using the MULTSESS userid and password
*    Wait for the TSO ‘READY’ message
*    Go into ISPF edit (option 2)
*    Return to ISPF primary menu
*    Go into browse and obtain a member list
*    Select a member from the member list
*    Split the screen at line 10
*    Select option 6 on bottom half and issue LISTC
*    Return to MULTSESS to display screen to the user.
*
* Wait for prompt for userid anywhere on line 1

 UNTIL 1,* ‘ENTER USERID -’
    WAIT
*
* Type TSO userid (same as MULTSESS userid)
*
  VAR   &SYSUSER
  ENTER
*
* Should prompt for password - if not error code 10
*
  JUMPIF 3,* ‘ENTER CURRENT PASSWORD’
     END 10
*
* Type TSO password (same as MULTSESS password)
*
  VAR    &SYSPASS
  ENTER
*
* If broadcast messages fill the screen (***), press ENTER
*
  WHILE *,2 ‘***’
     ENTER
*
* If not ‘READY’ then error code 20
*
  JUMPIF *,2 ‘READY’
     END 20
*
* Invoke ISPF option 2
*
  DATA ‘ISPF 2’
  ENTER
*
* Keep pressing PFK15 until back to ISPF primary menu
*
LABEL HIT-PFKEY-15
   PFKEY 15
   JUMPIF 1,27 ‘ISPF-PDF’
      GOTO HIT-PFKEY-15
*
* Select option 1 (Browse)
*
  DATA   ‘1’
  ENTER
*
* Type the partitioned dataset name to be browsed
*
  CURSOR 5,18
  DATA   ‘CAK0001’
  ERASE-EOF
  CURSOR 6,18
  DATA   ‘CKATV10’
  ERASE-EOF
  CURSOR 7,18
  DATA   ‘CNTL’
  ERASE-EOF
  ENTER
```

continued ...

```
*
* Select the second member from the list
*
  DOWN-CURSOR 3
  LEFT-CURSOR 19
  DATA   ‘S’
  ENTER
*
* Split the screen at line 10
*
  CURSOR 10,80
  PFKEY 14
*
* Select option 6 (TSO) on bottom half of screen
*
  DATA ‘6’
  ENTER
*
* Issue LISTC command on bottom half of screen
*
 DATA ‘LISTC’
 ENTER
*
* End of script - show the screen to the user
*
  WAIT
  END
```

Index

Reader's Comment Form

If you find any discrepancy in the information contained in this publication, please complete this form and mail it to the address below.

The authors may use, or distribute, any of the information you supply in any way they consider appropriate without incurring any obligation whatsoever.

Publication number PAT0001.012 - Twelth Edition (September 2001)

Please write your comments below and on the following page and return this form to the

Documentation Manager
Horton Manor
PassGo Technologies Ltd
Ilminster, Somerset
TA19 9PY
England

Reader's Comment Form